

# Betriebssysteme Projekt „Scheduling Simulator“

Nikolaj Pauly *Betriebssysteme*  
*Hochschule Ruhr West*  
Essen, Germany  
nikolaj.pauly@stud.hs-ruhrwest.de

Saliha Balci *Betriebssysteme*  
*Hochschule Ruhr West*  
Bottrop, Germany  
saliha.balci@stud.hs-ruhrwest.de

Andreas Benjamin Roscher *Betriebssysteme*  
*Hochschule Ruhr West*  
Oberhausen, Germany  
benjamin.roscher@stud.hs-ruhrwest.de

21. August 2023

## **Zusammenfassung**

Im Rahmen dieses Projekts soll eine Simulationsumgebung für Prozess-Scheduler entwickelt und implementiert werden. Das Ziel besteht darin, eine Application Programming Interface (API) zu schaffen, mit der Entwickler ihre Scheduling-Algorithmen in der Umgebung einfach implementieren und evaluieren können. Der Fokus liegt auf der Schaffung einer vollständig funktionsfähigen Kernsimulation. Das Projekt beinhaltet das Konzipieren eines Software-Konzepts, das Prozesse mithilfe des Discrete-Event-Ansatzes simuliert und Entwicklern Zugriff auf den Planungsmechanismus des DES ermöglicht. Jede Planung eines Prozesses im Simulator stellt dabei die Zuweisung von Rechenzeit einer CPU dar, wobei von einer Umgebung mit einer einzigen CPU ausgegangen wird. Zusätzlich sollen grundlegende Scheduling-Algorithmen, die in den Vorlesungen behandelt wurden, implementiert werden, um die Funktionalität des Simulators zu veranschaulichen.

Link zum Code: [Code auf GitHub](#)

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung und Motivation</b>	<b>2</b>
<b>2</b>	<b>Ziele</b>	<b>2</b>
<b>3</b>	<b>Grundlagen</b>	<b>3</b>
3.1	Prozesse und Threads . . . . .	3
3.1.1	Prozesse . . . . .	3
3.1.2	Threads . . . . .	3
3.2	First Come First Served Scheduler . . . . .	4
3.3	Shortest Job First Scheduler . . . . .	4
3.4	Prioritätsscheduling . . . . .	5
3.5	Round Robin Scheduling . . . . .	6
3.6	Round Robin mit Priorität Scheduling . . . . .	7
<b>4</b>	<b>Planung und Projektmanagement</b>	<b>8</b>
4.1	Arbeiten mit Git Bash und GitHub . . . . .	8
4.2	Zeitmanagment mit Gantt-Diagramm . . . . .	9
4.3	Projektmanagement mit Kanban . . . . .	9
4.4	Aufteilung der Aufgaben im Team . . . . .	9
<b>5</b>	<b>Simulationsprogramm für Scheduling</b>	<b>11</b>
5.1	Aufbau der Benutzeroberfläche . . . . .	11
<b>6</b>	<b>Entwicklung der Simulation</b>	<b>15</b>
6.1	Vereinfachungen und Annahmen . . . . .	15
6.2	Zeitstrahlen erstellen und verarbeiten . . . . .	15
<b>7</b>	<b>Unified Modeling Language (UML)</b>	<b>16</b>
<b>8</b>	<b>Download und erster Start</b>	<b>18</b>
<b>9</b>	<b>Fazit und Ausblick</b>	<b>19</b>

# 1 Einleitung und Motivation

Die effiziente Planung und Ausführung von Prozessen ist ein zentraler Aspekt moderner Betriebssysteme. Um die verschiedenen Scheduling-Algorithmen und deren Auswirkungen auf die Systemleistung besser zu verstehen, ist es von großer Bedeutung, eine Simulationsumgebung zu haben, in der diese Algorithmen getestet und verglichen werden können.

Das vorliegende Projekt hat zum Ziel, eine solche Simulationsumgebung für Prozess-Scheduler zu entwickeln und zu implementieren. Durch die Erstellung einer API ermöglicht das Projekt Entwicklern, ihre eigenen Scheduling-Algorithmen einfach in die Umgebung einzubetten und deren Leistung zu evaluieren.

Durch die Implementierung grundlegender Scheduling-Algorithmen, die in der Vorlesung behandelt wurden, kann die Funktionalität des Simulators verdeutlicht werden.

## 2 Ziele

Das Ziel dieses Projekts ist die Entwicklung und Implementierung einer Simulationsumgebung für Prozess-Scheduler. Das Projekt zielt darauf ab, Entwicklern eine API zur Verfügung zu stellen, die eine einfache Implementierung und Evaluation verschiedener Scheduling-Algorithmen innerhalb der Umgebung ermöglicht. Der Fokus liegt auf der Schaffung einer vollständig funktionsfähigen Kernsimulation. Das Projekt beinhaltet das Konzipieren eines Software-Konzepts, das den Discrete-Event-Ansatz verwendet, um Prozesse zu simulieren und Entwicklern Zugriff auf den Planungsmechanismus des DES zu gewähren. Darüber hinaus sollen grundlegende Scheduling-Algorithmen, die in den Vorlesungen behandelt wurden, implementiert werden, um die Funktionalität des Simulators zu demonstrieren.

## 3 Grundlagen

### 3.1 Prozesse und Threads

#### 3.1.1 Prozesse

Ein Prozess ist eine Instanz eines laufenden Programms auf einem Computer. Er wird als eine Einheit betrachtet, die Ressourcen wie Speicher, CPU-Zeit, Dateien und Eingabe-/Ausgabegeräte beansprucht. Jeder Prozess hat seinen eigenen Adressraum, der den Speicherbereich repräsentiert, in dem er ausgeführt wird, sowie seinen eigenen Satz von Ressourcen.

Prozesse werden typischerweise durch den Betriebssystemkern verwaltet. Beim Starten eines Programms erzeugt das Betriebssystem einen neuen Prozess, der den Programmcode, den Datenbereich, die Stapel und andere Informationen enthält. Jeder Prozess erhält eine eindeutige Prozess-ID (PID), mit der er identifiziert werden kann.

Prozesse werden in der Regel als unabhängige Entitäten betrachtet, da sie ihren eigenen Speicher und ihre eigenen Ressourcen haben. Sie kommunizieren normalerweise über Mechanismen wie Interprozesskommunikation (IPC), um Daten auszutauschen oder Aufgaben gemeinsam zu erledigen. [1]

#### 3.1.2 Threads

Ein Thread ist eine Ausführungseinheit innerhalb eines Prozesses. Ein Prozess kann aus einem oder mehreren Threads bestehen. Jeder Thread repräsentiert eine separate Ausführungssequenz, die unabhängig von anderen Threads innerhalb desselben Prozesses abläuft. Threads teilen sich den Adressraum und die Ressourcen des Prozesses, was bedeutet, dass sie auf denselben Speicherbereich, dieselben Dateien und dieselben Eingabe-/Ausgabegeräte zugreifen können.

Threads ermöglichen paralleles oder asynchrones Arbeiten innerhalb eines Prozesses. Mehrere Threads können gleichzeitig ausgeführt werden, wodurch die Effizienz und die Möglichkeiten der Programmierung verbessert werden. Jeder Thread hat seinen eigenen Programmzähler, seinen eigenen Stapel und seinen eigenen Zustand, aber sie teilen sich den globalen Speicherbereich des Prozesses.

Threads innerhalb eines Prozesses können sich gegenseitig beeinflussen und Informationen austauschen, indem sie auf denselben Speicherbereich zugreifen. Dies erfordert jedoch eine sorgfältige Synchronisation, um Dateninkonsistenzen und Wettlaufsituationen zu vermeiden.

Im Allgemeinen bieten Threads eine effiziente Möglichkeit, Aufgaben in einem Programm zu parallelisieren und die Leistung zu verbessern. Sie können zur gleichzeitigen Ausführung von Berechnungen, zur asynchronen Verarbeitung von Eingabe-/Ausgabevorgängen oder zur Optimierung von Ressourcennutzung und Reaktionsfähigkeit eingesetzt werden.

Es ist wichtig zu beachten, dass Threads innerhalb desselben Prozesses denselben Kontext teilen und daher die sorgfältige Verwaltung von Ressourcen und die Synchronisierung bei der gemeinsamen Nutzung von Daten erforderlich sind, um potenzielle Probleme wie Rennbedingungen oder Deadlocks zu vermeiden. [1]

### 3.2 First Come First Served Scheduler

Der First Come First Served (FCFS) Scheduler ist ein einfacher Scheduling-Algorithmus, der in Betriebssystemen verwendet wird, um die Reihenfolge der Ausführung von Prozessen zu bestimmen. Bei diesem Algorithmus wird jedem eintreffenden Prozess die CPU-Zeit in der Reihenfolge zugewiesen, in der sie in die Warteschlange eingetreten sind. Das bedeutet, dass der zuerst eintreffende Prozess als erster ausgeführt wird und die CPU so lange verwendet, bis er seine Ausführung beendet oder blockiert wird.

Die Arbeitsweise des FCFS-Schedulers ist relativ einfach. Wenn ein Prozess eintrifft, wird er in die Warteschlange der bereiten Prozesse eingereiht. Wenn die CPU frei ist, wird der Prozess aus der Warteschlange genommen und beginnt seine Ausführung. Der Prozess behält die CPU so lange, bis er entweder seine Ausführung vollständig abgeschlossen hat oder blockiert wird, z.B. wenn er auf eine Eingabe/Ausgabe-Operation wartet. In diesem Fall wird der Prozess aus der CPU entfernt und der nächste Prozess in der Warteschlange wird ausgeführt.

Der FCFS-Scheduler funktioniert gut, wenn die Prozesse ähnliche Ausführungszeiten haben und es keine speziellen Anforderungen oder Prioritäten gibt. Allerdings kann dieser Algorithmus zu sogenanntem "Wartezeit-Verhungern" führen. Das bedeutet, dass Prozesse mit längeren Ausführungszeiten, die später in die Warteschlange eingetreten sind, länger auf ihre Ausführung warten müssen, während kürzere Prozesse, die früher eingetroffen sind, die CPU länger belegen. Dadurch kann die Effizienz des Systems beeinträchtigt werden, insbesondere wenn lang andauernde Prozesse die Ausführung kurzlebiger Prozesse blockieren.

Es ist wichtig zu beachten, dass der FCFS-Scheduler ein nicht-präemptiver Algorithmus ist, was bedeutet, dass ein Prozess, sobald er die CPU in Besitz genommen hat, nicht unterbrochen oder vorzeitig beendet wird. Dies kann zu einer niedrigen CPU-Auslastung führen, wenn ein langer Prozess die CPU blockiert und andere Prozesse in der Warteschlange auf ihre Ausführung warten müssen.

Insgesamt ist der FCFS-Scheduler ein einfacher, aber nicht unbedingt effizienter Algorithmus, der oft in einfachen Betriebssystemen oder in Situationen verwendet wird, in denen die Ausführungszeiten der Prozesse vorhersehbar sind und keine Priorisierung erforderlich ist. [1]

### 3.3 Shortest Job First Scheduler

Der Shortest Job First (SJF) Scheduler ist ein Scheduling-Algorithmus, der in Betriebssystemen verwendet wird, um die Reihenfolge der Ausführung von Prozessen zu bestimmen. Bei diesem Algorithmus wird jedem eintreffenden Prozess die CPU-Zeit basierend auf seiner geschätzten oder gemessenen Ausführungsdauer zugewiesen. Der Prozess mit der kürzesten erwarteten Ausführungszeit erhält die höchste Priorität und wird als erster ausgeführt.

Die Arbeitsweise des SJF-Schedulers besteht darin, die Prozesse in die Warteschlange einzufügen, basierend auf ihrer erwarteten Ausführungsdauer. Wenn ein Prozess eintrifft, wird seine Ausführungsdauer mit den verbleibenden Ausführungszeiten der bereits in der Warteschlange befindlichen Prozesse verglichen. Der Prozess mit der kürzesten verbleibenden Ausführungszeit wird als nächstes ausgeführt. Auf diese Weise wird versucht, die Prozesse mit den kürzesten Ausführungszeiten priorisiert auszuführen, um die Gesamtausführungszeit zu minimieren.

Der SJF-Scheduler kann in zwei Varianten auftreten: nicht-präemptiv und präemptiv. Im nicht-präemptiven SJF-Scheduling-Modus wird der ausgewählte Prozess die CPU behalten, bis er seine Ausführung vollständig abgeschlossen hat oder blockiert wird. Im präemptiven SJF-Scheduling-Modus kann ein Prozess unterbrochen werden, wenn ein neuer Prozess mit einer noch kürzeren Ausführungsdauer eintrifft. In diesem Fall wird der neue Prozess priorisiert und die Ausführung des aktuellen Prozesses wird unterbrochen, um den kürzeren Prozess auszuführen.

Der SJF-Scheduler ist besonders effizient in Situationen, in denen die Ausführungszeiten der Prozesse bekannt oder gut geschätzt sind. Durch die Priorisierung der kürzesten Prozesse wird die Durchschnittswartezeit der Prozesse reduziert und die Gesamtausführungszeit des Systems optimiert. Dadurch kann der SJF-Scheduler eine hohe Systemleistung und eine effiziente Ressourcennutzung erreichen.

Es gibt jedoch auch potenzielle Nachteile des SJF-Schedulers. Wenn die Ausführungszeiten der Prozesse nicht genau bekannt sind, kann es zu einer Fehleinschätzung der Ausführungsdauer kommen, was zu einer inkorrekten Priorisierung der Prozesse führen kann. Darüber hinaus kann der SJF-Algorithmus zu einer Vernachlässigung längerer Prozesse führen, was als "Verhungern" bezeichnet wird, ähnlich wie beim FCFS-Scheduler.

Insgesamt ist der SJF-Scheduler ein effizienter Algorithmus, der in Situationen, in denen die Ausführungszeiten der Prozesse bekannt sind oder gut geschätzt werden können, gute Ergebnisse liefert. Es ist jedoch wichtig, die Eigenschaften der Prozesse und die Schätzgenauigkeit der Ausführungszeiten zu berücksichtigen, um die Effizienz des Schedulers zu maximieren. [1]

### 3.4 Prioritätsscheduling

Das Prioritätsscheduling ist ein Scheduling-Algorithmus, der in Betriebssystemen verwendet wird, um die Ausführungsreihenfolge von Prozessen basierend auf ihrer Priorität zu bestimmen. Jeder Prozess wird mit einer Priorität versehen, die entweder statisch oder dynamisch sein kann. Ein Prozess mit einer höheren Priorität hat Vorrang vor einem Prozess mit einer niedrigeren Priorität und wird vorrangig ausgeführt.

Beim Prioritätsscheduling wird jedem Prozess bei seiner Erstellung eine Priorität zugewiesen. Die Prioritäten können auf verschiedene Arten zugewiesen werden, zum Beispiel basierend auf der Art des Prozesses, der Wichtigkeit der Aufgabe oder anderen Faktoren. Prozesse mit höherer Priorität werden bevorzugt und haben eine größere Chance, die CPU zu erhalten. Wenn mehrere Prozesse mit derselben Priorität bereit sind, wird oft das FCFS-Prinzip angewendet, d.h., der zuerst eingetroffene Prozess mit der höchsten Priorität wird als erster ausgeführt.

Es gibt zwei Arten von Prioritätsscheduling: nicht-präemptiv und präemptiv. Im nicht-präemptiven Prioritätsscheduling behält ein Prozess die CPU, bis er seine Ausführung vollständig abgeschlossen hat oder blockiert wird. Dies bedeutet, dass ein Prozess mit hoher Priorität die CPU dauerhaft beanspruchen kann, selbst wenn ein Prozess mit noch höherer Priorität eintreffen sollte. Im präemptiven Prioritätsscheduling hingegen kann ein Prozess mit niedrigerer Priorität unterbrochen werden, wenn ein Prozess mit höherer Priorität eintrifft. Der Prozess mit der höheren Priorität wird dann vorrangig ausgeführt, um eine bessere Ausnutzung der Ressourcen zu gewährleisten.

Das Prioritätsscheduling bietet Flexibilität und ermöglicht die Priorisierung von Prozessen basierend auf ihren Anforderungen oder Wichtigkeiten. Es ist jedoch wichtig, die Prioritäten sorgfältig zu verwalten, um sicherzustellen, dass Prozesse mit niedrigerer

Priorität nicht dauerhaft vernachlässigt werden, was zu sogenanntem "Verhungern" führen kann. Um dies zu vermeiden, kann ein Verfahren wie das Aging verwendet werden, bei dem die Prioritäten von Prozessen mit der Zeit erhöht werden, um sicherzustellen, dass auch Prozesse mit niedrigerer Priorität gelegentlich ausgeführt werden.

Ein potenzieller Nachteil des Prioritätsschedulings ist die Möglichkeit von Inversionen der Priorität. Wenn ein Prozess mit niedrigerer Priorität eine gemeinsame Ressource blockiert, kann ein Prozess mit höherer Priorität auf die Freigabe dieser Ressource warten und dadurch in seiner Ausführung verzögert werden. Dies kann zu unerwarteten Verzögerungen führen und die Effizienz des Systems beeinträchtigen.

Insgesamt ermöglicht das Prioritätsscheduling eine flexible Zuweisung von Ressourcen basierend auf der Priorität der Prozesse. Es ist wichtig, die Prioritäten sorgfältig zu verwalten und mögliche Probleme wie Verhungern und Prioritätsinversionen zu berücksichtigen. [1]

### 3.5 Round Robin Scheduling

Das Round-Robin-Scheduling ist ein weit verbreiteter Scheduling-Algorithmus, der in Betriebssystemen eingesetzt wird, um die CPU-Zeit gerecht zwischen mehreren Prozessen aufzuteilen. Bei diesem Algorithmus erhalten alle Prozesse in der Warteschlange regelmäßig und zyklisch Zeit auf der CPU, unabhängig von ihrer Priorität oder Ausführungsdauer.

Die Arbeitsweise des Round-Robin-Schedulings besteht darin, jedem Prozess eine festgelegte Zeitscheibe oder auch Quantum genannt zuzuweisen. Diese Zeitscheibe definiert die maximale Zeit, die ein Prozess auf der CPU verbringen darf, bevor er unterbrochen und ein anderer Prozess ausgeführt wird. Die Prozesse werden in einer Warteschlange organisiert, oft als zirkuläre Liste, und der Algorithmus durchläuft diese Liste in einer Runde (daher der Name "Round Robin").

Wenn ein Prozess die CPU betritt und seine zugewiesene Zeitscheibe noch nicht abgelaufen ist, wird er ausgeführt. Wenn die Zeitscheibe abläuft, wird der Prozess unterbrochen und in die Warteschlange zurückgestellt. Der nächste Prozess in der Warteschlange wird dann ausgewählt und für die nächste Zeitscheibe auf der CPU platziert. Dieser Vorgang wird fortgesetzt, bis alle Prozesse ihre Ausführung abgeschlossen haben.

Der Round-Robin-Algorithmus bietet Fairness, da jeder Prozess eine gleiche Chance erhält, die CPU zu nutzen. Er sorgt auch dafür, dass kein Prozess die CPU für eine unangemessen lange Zeit blockiert, da die Zeitscheiben aufgeteilt werden. Dies ist besonders wichtig in Multiuser- oder Multitasking-Umgebungen, in denen mehrere Benutzer oder Anwendungen gleichzeitig ausgeführt werden.

Ein potenzieller Nachteil des Round-Robin-Schedulings liegt in der Effizienz bei langen Prozessen. Wenn ein Prozess eine längere Ausführungszeit hat und die Zeitscheibe klein ist, kann es zu einer hohen Anzahl von Kontextwechseln kommen, was zu einem Overhead führt und die Gesamtausführungszeit erhöht. Um dieses Problem zu mildern, kann die Größe der Zeitscheibe optimiert werden, um eine gute Balance zwischen Fairness und Effizienz zu erreichen.

Darüber hinaus kann das Round-Robin-Scheduling zu einem Phänomen führen, das als "Latenzzeit" bezeichnet wird. Bei diesem Phänomen kann ein Prozess, der kurz vor Ende seiner Zeitscheibe unterbrochen wurde, bei seiner erneuten Ausführung länger brauchen, um die ursprüngliche Arbeit fortzusetzen, da sein interner Zustand möglicherweise nicht mehr im Cache liegt. Dies kann zu einer geringeren Effizienz führen, insbesondere bei



rechenintensiven Aufgaben.

Insgesamt bietet das Round-Robin-Scheduling eine ausgewogene Verteilung der CPU-Zeit zwischen Prozessen und gewährleistet Fairness. Es eignet sich gut für Umgebungen mit vielen gleichberechtigten Prozessen oder wenn es wichtig ist, eine reaktionsschnelle Benutzerinteraktion sicherzustellen. [1]

### **3.6 Round Robin mit Priorität Scheduling**

Round Robin mit Priorität ist eine Variante des Round-Robin-Schedulings, bei der den Prozessen zusätzlich zu ihrer Position in der zyklischen Warteschlange auch Prioritäten zugewiesen werden. Dadurch wird die Ausführungsreihenfolge der Prozesse basierend auf ihrer Priorität bestimmt, während das Round-Robin-Verfahren verwendet wird, um innerhalb der Prioritätsgruppen eine faire Verteilung der CPU-Zeit sicherzustellen.

Die Arbeitsweise des Round Robin mit Priorität-Schedulings besteht darin, jedem Prozess eine Priorität zuzuweisen, die seine Wichtigkeit oder Dringlichkeit widerspiegelt. Prozesse mit höherer Priorität werden bevorzugt behandelt und haben eine größere Chance, die CPU zu erhalten. Innerhalb einer Prioritätsgruppe wird das Round-Robin-Verfahren angewendet, um sicherzustellen, dass die Prozesse in fairer Weise die CPU-Zeit teilen.

Die Prozesse werden in der Reihenfolge ihrer Prioritäten in Prioritätsschlangen organisiert. Innerhalb jeder Prioritätsschlange wird das Round-Robin-Scheduling angewendet, um jedem Prozess eine bestimmte Zeitscheibe auf der CPU zuzuweisen. Wenn die Zeitscheibe abläuft, wird der Prozess in die Warteschlange seiner Priorität zurückgestellt, und der nächste Prozess in derselben Prioritätsschlange erhält die CPU. Wenn alle Prozesse einer Prioritätsschlange ihre Zeitscheiben aufgebraucht haben, wechselt der Scheduler zur nächsten Prioritätsschlange mit höherer Priorität und führt das Round-Robin-Scheduling dort fort.

Das Round Robin mit Priorität-Scheduling bietet eine Kombination aus Fairness und Priorisierung. Prozesse mit höherer Priorität erhalten einen größeren Anteil an der CPU-Zeit und werden bevorzugt ausgeführt, während das Round-Robin-Verfahren dafür sorgt, dass kein Prozess die CPU dauerhaft blockiert und die Ausführungszeit gerecht aufgeteilt wird. Dadurch werden sowohl die Reaktionsfähigkeit für wichtigere Aufgaben als auch die Ausführung von weniger wichtigen Aufgaben gewährleistet.

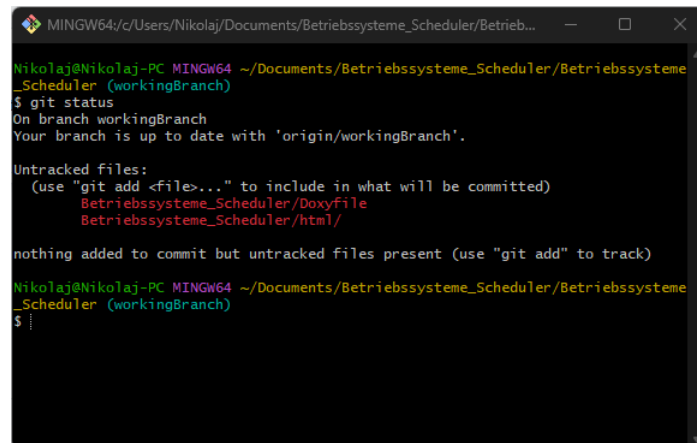
Bei der Implementierung des Round Robin mit Priorität-Schedulings ist es wichtig, die Prioritäten sorgfältig zu verwalten und mögliche Probleme wie Verhungern zu berücksichtigen. Es können Mechanismen wie die Anpassung der Prioritäten im Laufe der Zeit (Aging) verwendet werden, um sicherzustellen, dass Prozesse mit niedrigerer Priorität gelegentlich ausgeführt werden und nicht dauerhaft vernachlässigt werden.

Insgesamt ermöglicht das Round Robin mit Priorität-Scheduling eine ausgewogene Verteilung der CPU-Zeit unter Berücksichtigung der Prioritäten der Prozesse. Es ist eine effektive Methode, um Priorisierung und Fairness zu kombinieren und eignet sich gut für Situationen, in denen unterschiedliche Aufgaben unterschiedliche Prioritäten haben und dennoch eine faire Verteilung der Ressourcen gewährleistet sein soll. [1]

## 4 Planung und Projektmanagement

### 4.1 Arbeiten mit Git Bash und GitHub

Git Bash ist eine Kommandozeilenschnittstelle (Command Line Interface (CLI)), die auf Git basiert, einem verteilten Versionskontrollsystem. Es bietet eine konsolenbasierte Umgebung für die Verwaltung von Git-Repositories. Mit Git Bash können Entwickler Befehle wie "clone", "commit", "push" und "pull" ausführen, um Codeänderungen zu verfolgen, zu verwalten und mit anderen Teammitgliedern zu teilen.



```
MINGW64/c/Users/Nikolaj/Documents/Betriebssysteme_Scheduler/Betrieb...
Nikolaj@Nikolaj-PC MINGW64 ~/Documents/Betriebssysteme_Scheduler/Betriebssysteme_Scheduler (workingBranch)
$ git status
On branch workingBranch
Your branch is up to date with 'origin/workingBranch'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Betriebssysteme_Scheduler/Doxyfile
    Betriebssysteme_Scheduler/html/

nothing added to commit but untracked files present (use "git add" to track)
Nikolaj@Nikolaj-PC MINGW64 ~/Documents/Betriebssysteme_Scheduler/Betriebssysteme_Scheduler (workingBranch)
$
```

Abbildung 1: GitBash mit Status Aufruf

GitHub ist eine webbasierte Plattform für die Versionskontrolle und Zusammenarbeit von Codeprojekten. Es ermöglicht Entwicklern, Git-Repositories hochzuladen und gemeinsam an Code zu arbeiten. GitHub bietet Funktionen wie Pull Requests, ein Kanban Board für Projekte, welche die Zusammenarbeit innerhalb eines Projekts zu erleichtern.

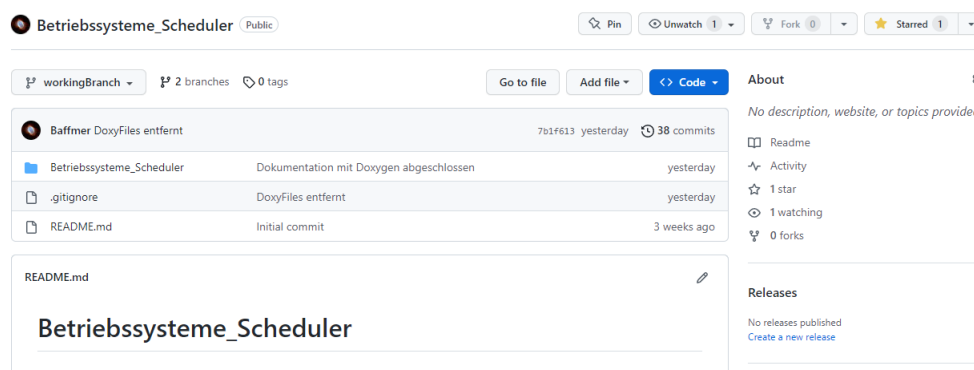


Abbildung 2: Ansicht auf GitHub für das Repository

Die Kombination von Git Bash und GitHub bietet Entwicklern eine leistungsstarke Lösung für die Versionskontrolle und Zusammenarbeit. Mit Git Bash kann man Git-Befehle über die Kommandozeile ausführen und auf die Funktionen von GitHub zugreifen, um Code zu veröffentlichen, zu verwalten und mit anderen Entwicklern zusammenzuarbeiten. Diese Tools sind in der Softwareentwicklung weit verbreitet und erleichtern die effektive Verwaltung von Codeprojekten.

## 4.2 Zeitmanagement mit Gantt-Diagramm

Der zeitliche Ablauf des Projekts wurde mit einem Gantt-Diagramm geplant. Der Vorteil des Gantt-Diagramms ist, dass hier die Tasks der Entwickler in Phasen eingeteilt werden kann. Dies ist notwendig, da manche Tasks auf anderen aufbauen. Das Gantt-Diagramm wurde in einer Excel-Datei erstellt, da die Anzahl der Tasks doch recht umfangreich und eine Darstellung hier in der Dokumentation nicht möglich ist.

## 4.3 Projektmanagement mit Kanban

Kanban ist eine agile Projektmanagementmethode, die auf einem visuellen Board basiert. Aufgaben werden als Tasks dargestellt und durch verschiedene Phasen des Arbeitsprozesses bewegt. Durch Begrenzung der gleichzeitig bearbeiteten Aufgaben und kontinuierliche Verbesserung wird die Produktivität gesteigert. Kanban eignet sich für Projekte mit sich ändernden Anforderungen und fördert Transparenz, Effizienz und Teamarbeit.

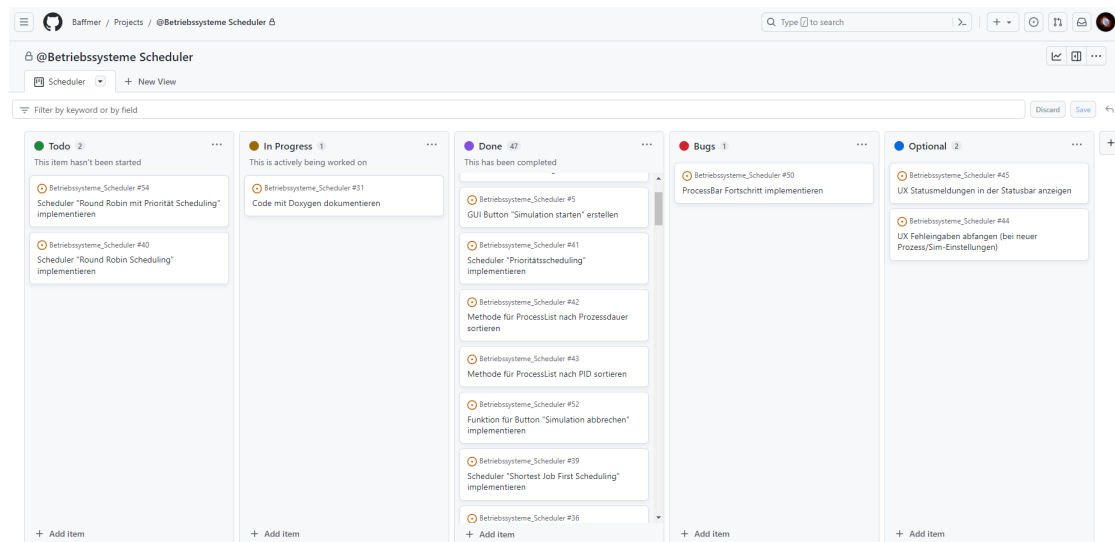


Abbildung 3: Ansicht des Kanban Boards

In Abbildung 3 erkennt man die Tasks in den verschiedenen Phasen.

## 4.4 Aufteilung der Aufgaben im Team

Zunächst mussten die Aufgaben bzw. die Planung der Aufgaben diskutiert und erstellt werden. Dafür wurde das Kanban Board vom Projektmanager auf GitHub verwendet, welches man in Abbildung 3 erkennen kann. Zunächst wurden hauptsächlich GUI Tasks erstellt. Sukzessive kamen dann neue Tasks hinzu, so dass das Board nach und nach die Elemente wie in der nachfolgenden Tabelle 1 zu erkennen ist. Ebenfalls ist aufgelistet welcher Entwickler welche Tasks übernommen hat.

Tätigkeiten	Entwickler		
	Nikolaj	Saliha	Andreas
Oberklasse „Scheduler“ für die Scheduler erstellen	X		
Scheduler „Round Robin Scheduling“ implementieren		X	
ProcessBar Fortschritt implementieren			X
Scheduler „Round Robin mit Priorität Scheduling“ implementieren	X		
Code mit Doxygen dokumentieren		X	
UX Symbole für Starten / Pausieren / Abbrechen / Einstellungen	X		
GUI Button „Simulation starten“ erstellen			X
Scheduler „Prioritätsscheduling“ implementieren			X
Methode für ProcessList nach Prozessdauer sortieren		X	
Methode für ProcessList nach PID sortieren		X	
Funktion für Button „Simulation abbrechen“ implementieren	X		
Scheduler „Shortest Job First Scheduling“ implementieren		X	
Prozess „Zeitstrahlen“ erstellen	X		
Evaluation Simulationsdauer messen und ausgeben			X
Evaluation Anzahl der Prozesswechsel zählen und ausgeben			X
Scheduler „First Come First Served“ implementieren	X		
Funktion für Button Prozess abbrechen implementieren		X	
GUI Eingabefeld für die I/O Dauer erstellen		X	
GUI Eingabefeld für die Dauer eines Prozesswechsels erstellen	X		
GUI Beschreibungen für die verschiedenen Scheduler hinzufügen			X
GUI Eingabe für die Simulationsgeschwindigkeit hinzufügen		X	
GUI Eingabe für das Zeit-Quantum erstellen			X
Hilfe Kategorie „Hilfe“ implementieren	X		
GUI Button „Simulationparameter einstellen“ erstellen		X	
Hintergrundfarben für Zeilen in der Prozessliste implementieren			X
Hilfe Kategorie „Über...“ implementieren	X		
Prozess Beispiele erstellen beim Klicken auf den Button „Beispiele laden“			X
Variable vom Dropdown Menü Scheduler einlesen	X		
GUI Statusbar erstellen			X
Prozess Tabelle aktualisieren bei erstellen/bearbeiten/löschen von Prozessen		X	
Klasse „Prozess“ erstellen			X
Funktion für Button „Prozess löschen“ implementieren		X	
Methode für ProcessList nach Prio sortieren			X
Singleton Pattern für Prozesse erstellen		X	
GUI Button „Prozess abbrechen“ erstellen		X	
GUI Eingabefenster für „Prozess erstellen“ erstellen	X		
GUI Dropdown Menü „Scheduling Methode“ erstellen	X		
Funktion für Button Simulation pausieren implementieren			X
GUI Button „Simulation pausieren“ erstellen	X		
Funktion für Button „Prozess bearbeiten“ implementieren		X	
GUI Button „Prozess bearbeiten“ erstellen	X		
GUI Button „Simulation abbrechen“ erstellen	X		
GUI Label „Anzahl Prozesswechsel“ erstellen			X
GUI Button „Prozess löschen“ erstellen		X	
GUI Button „Prozess erstellen“ erstellen		X	
GUI Button „Beispiele laden“ erstellen			X
Prozess-Info-Tabelle aktualisieren bei Auswahl eines Prozesses		X	
GUI Prozess Tabelle erstellen	X		
GUI Prozess Details Tabelle erstellen		X	
GUI Layout anpassen	X		
GUI Label „Simulationszeit“ erstellen			X
UX Statusmeldungen in der Statusbar anzeigen		X	
UX Fehleingaben abfangen (bei neuer Prozess/Sim-Einstellungen)			X

Tabelle 1: Aufgabenverteilung

## 5 Simulationsprogramm für Scheduling

### 5.1 Aufbau der Benutzeroberfläche

Die Oberfläche des Hauptprogramms ist unterteilt in mehrere Bereiche, siehe Abbildung 4.

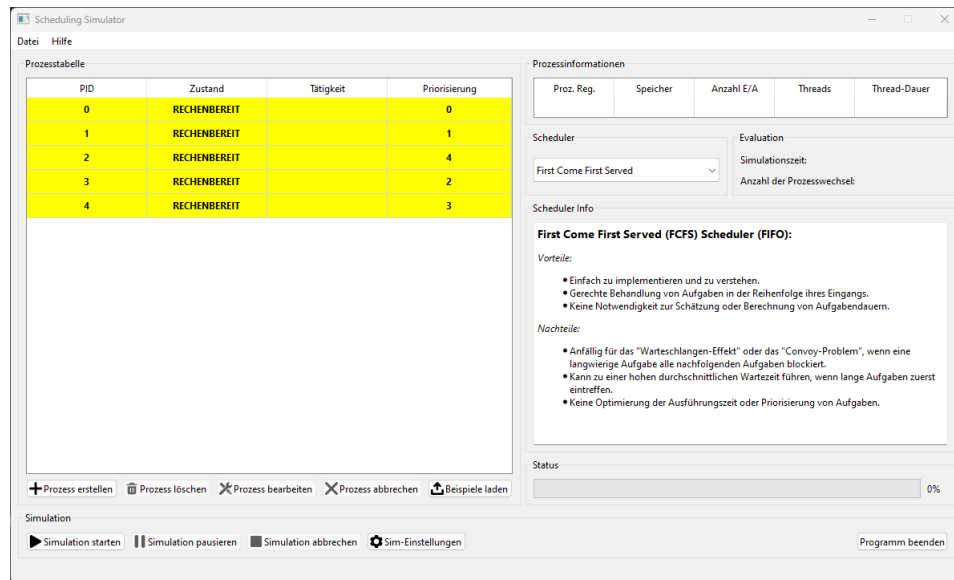


Abbildung 4: GUI des Hauptprogramms

Auf der linken Seite sieht man die „Prozestabelle“, in welcher alle Prozesse angezeigt werden. Prozesse können diverse Zustände einnehmen:

1. RECHNEND (grün)
2. RECHENBEREIT (gelb)
3. ABGESCHLOSSEN (grau)
4. ABGEBROCHEN (dunkel grau)
5. BLOCKIERT (rot)

Unterhalb dieser Tabelle finden sich die Buttons zum Erstellen, Löschen, Bearbeiten und Abbrechen eines Prozesses. Möchte der Anwender schnell Prozesse hinzufügen, so kann er dies mit einen Klick auf den Button „Beispiele laden“ tun. Es werden dann fünf Prozesse geladen, wie in Abbildung 4 zu sehen ist.

Im unteren Bereich der Oberfläche sind die Buttons bezüglich der Simulation zu erkennen. Hier kann eine Simulation gestartet, pausiert oder abgebrochen werden. Man beachte, dass eine Simulation nur pausiert oder abgebrochen werden kann, wenn zuvor eine gestartet wurde. In Abbildung 5 ist eine Benutzeroberfläche mit einer laufenden Simulation dargestellt.

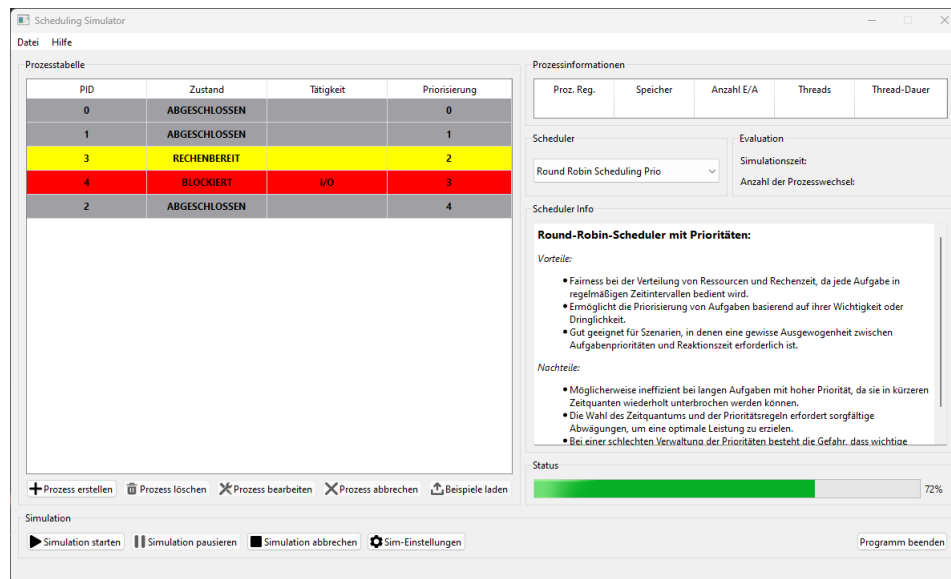


Abbildung 5: GUI des Hauptprogramms während einer Simulation

Auf der rechten Seite ist oben eine Tabelle mit Prozessinformationen platziert. Wird aus der Prozesstabelle mit einem Mausklick ein Prozess markiert, erhält man in dieser Tabelle nähere Informationen zum Prozess:

1. Prozessregister (Platzhalter)
2. Hauptspeicher (Platzhalter)
3. Anzahl Ein- u. Ausgaben
4. Anzahl der Threads
5. Dauer eines Threads

Darunter befindet sich die Auswahl der Scheduler in einem Drop-Down Menü:

1. FCFS
2. SJF
3. Prioritätsscheduling
4. Round Robin
5. Round Robin mit Priorität

Rechts daneben befindet sich Evaluationsbereich. Dieser gibt nach einer Simulation die Anzahl der Prozesswechsel sowie die Dauer der Simulation aus.

Unter der Schedulerauswahl und der Simulation ist ein Infofenster platziert, welcher die Vor- u. Nachteile des ausgewählten Scheduler darstellt.

Unten rechts im Hauptfenster befindet eine Progress-Bar, welche den Fortschritt der Simulation anzeigt.

Das Programm kann mit einem Klick auf „Programm beenden“, oder „Datei dann Beenden“ beendet werden.

Neue Prozesse können mit Klick auf den Button „Prozess erstellen“ erstellt werden. Klickt man auf diesen Button, so öffnet sich das Fenster wie in Abbildung 6. Hier kann man folgende Parameter einstellen:

1. Priorisierung
2. Prozessregister (Platzhalter)
3. Hauptspeicher (Platzhalter)
4. Anzahl Ein- u. Ausgaben
5. Anzahl der Threads
6. Dauer eines Threads

Man beachte, dass die PID automatisch, in fortschreitender Nummer, angelegt wird.

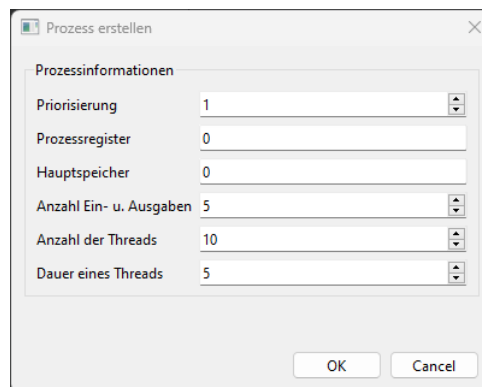


Abbildung 6: Fenster zum Erstellen eines neuen Prozesses

Neben diesen Buttons gibt es noch die Möglichkeit Simulationseinstellungen vorzunehmen. Dafür kann auf den Button „Sim-Einstellungen“ geklickt werden, was anschließend das Fenster, wie in Abbildung 7 zu sehen, öffnet. Hier können folgende Parameter eingestellt werden:

1. Dauer Zeit-Quantum (wird für die Round Robin Scheduler benötigt)
2. Dauer einer Input/Output Operation
3. Dauer eines Prozesswechsels
4. Simulations-Geschwindigkeit (langsam, normal, schnell)

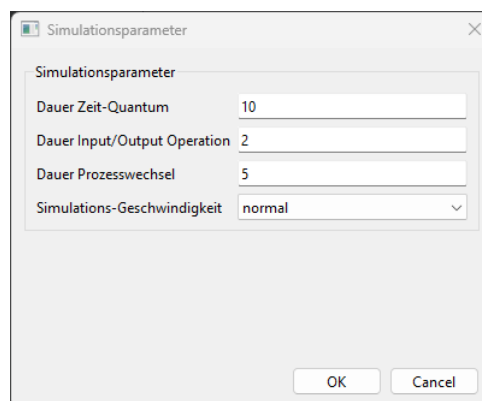


Abbildung 7: Fenster zum Erstellen der Simulationsparameter

Klickt man auf das Menü „Hilfe dann Über...“, so öffnet sich ein Infofenster wie in Abbildung 8 gezeigt wird. Hier finden sich Informationen wie eine Kurzbeschreibung über das Programm, ein Link zum GitHub-Repository und die Namen der Entwickler.

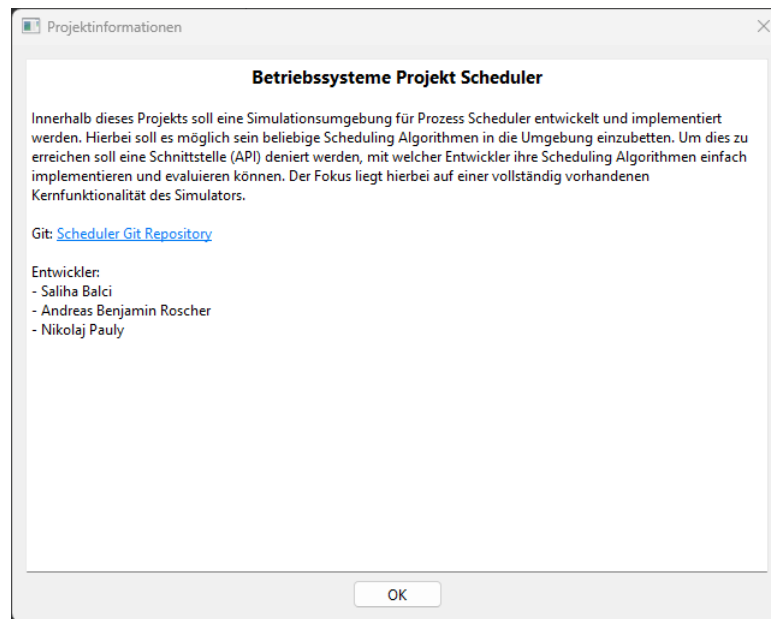


Abbildung 8: Projektinformationen



## 6 Entwicklung der Simulation

### 6.1 Vereinfachungen und Annahmen

Ein Prozess kann sehr kompliziert sein und eine große Anzahl von Parametern besitzen. Um die Simulation für den Anwender nicht zu kompliziert zu gestalten, werden folgende Bedingungen angenommen:

1. Alle I/O Operationen bekommen dieselbe Zeitdauer. Diese kann unter den Simulationsparametern eingestellt werden. Die Einstellung gilt dann aber für alle Prozesse.
2. Die Anzahl der I/O Operationen wird gleichmäßig über den ganzen Prozess verteilt.
3. Innerhalb eines Prozesses haben alle Threads dieselbe Zeitdauer. Diese kann bei der Erstellung eines Prozesses eingestellt werden. Die Einstellung gilt dann aber für alle Threads innerhalb des Prozesses.
4. Die Zeit wird nicht in Sekunden gemessen, sondern in dimensionslosen Zeitslots.

### 6.2 Zeitstrahlen erstellen und verarbeiten

Um einen Prozess zeitlich und visuell darstellen zu können, werden in der Simulation „Zeitstrahlen“ verwendet. Diese werden zusammen mit dem Prozess und seinen Parametern erstellt. Als Beispiel kann hier Abbildung 9 betrachtet werden. In diesem Beispiel wurden folgende Parameter für den Prozess verwendet:

1. 3 Prozesse (rot/grün/gelb)
2. Prozessdauer beträgt 3
3. Dauer einer I/O Operation beträgt 1
4. Anzahl der I/O Operationen beträgt 4

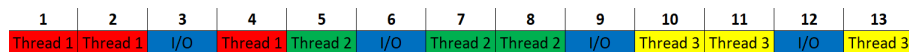


Abbildung 9: Zeitstrahl

Erstellung eines Zeitstrahls:

1. Füge die entsprechende Anzahl Threads mit der eingestellten Länge einer Liste hinzu.
2. Verteile die eingestellte Anzahl an I/O Operationen gleichmäßig über den Zeitstrahl.

Anschließend kann der Zeitstrahl der Reihe nach abgearbeitet werden. Im Programm wird dann einfach die Liste vom Index 0 an bis zur letzten Position abgearbeitet. Dies wird für alle erstellten Prozesse durchgeführt.

## 7 UML

Für eine bessere Klassenübersicht des Projekts, folgen hier UML Diagramme. Zunächst ein UML Diagramm für das Singleton „ProcessTable“, welches eine Liste aus „Processes“ enthält.



Abbildung 10: UML

Anschließend ein UML Diagramm für das „MainWindow“, welches die Scheduler instanziiert.

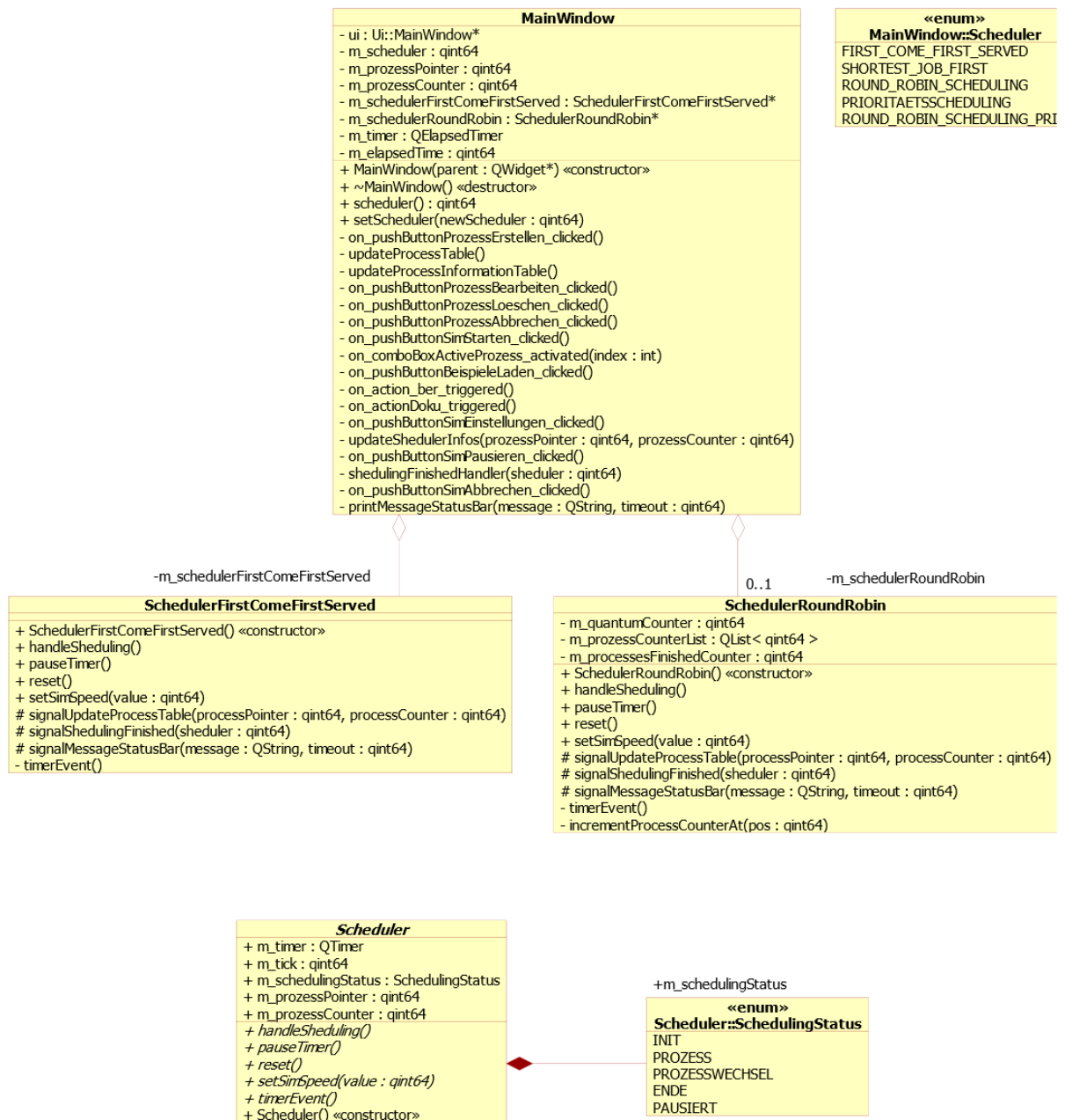


Abbildung 11: UML

## 8 Download und erster Start

Das Projekt „Scheduler Simulator“ wurde in C++ entwickelt. Da eine Benutzeroberfläche mit entwickelt werden sollte, wurde sich für dieses Projekt für die IDE „Qt Creator“ entschieden. Somit konnte die Benutzeroberfläche mit einem visuellen Editor entwickelt und auch getestet werden. Die IDE „Qt Creator“ ermöglicht das Debuggen in Entwickeln von Code in C++.

Da für das Programm keine Installationsdatei, bzw. eine ausführbare Datei erstellt wurde (was auf den meisten Systemen zu Problemen mit Schadsoftwareprogrammen führen würde), ist es erforderlich, sich für das Testen der Simulation die IDE auf der Homepage von „Qt Creator“ herunterzuladen:

Qt Open Source.

Hier muss darauf geachtet werden die „Qt Creator“ Version 6.5 zu installieren.

Anschließend muss ein Programm für Git installiert werden, um das Repository von GitHub herunter zu laden. Ein Programm, welches einfach und schlank ist, könnte Git Bash sein:

Git Bash für Windows.

Für die Installation des Programms können die Standardeinstellungen verwendet werden.

Nun kann das Projekt von GitHub geklont werden. Dafür muss im Wunschverzeichnis Git Bash in der Konsole geöffnet werden (ist Git Bash installiert, kann mit einem rechten Mausklick im Wunschverzeichnis „Git Bash Here“ ausgewählt werden).

Folgender Befehl lädt das Projekt in dieses Verzeichnis:

```
„git clone https://github.com/Baffmer/Betriebssysteme_Scheduler.git“
```

Nun kann „Qt Creator“ gestartet werden. Jetzt kann unter „Projekt öffnen“ in das Wunschverzeichnis navigiert werden. Zum Laden des Projekts wählt man folgende Datei aus:

```
„Betriebssysteme_Scheduler\Betriebssysteme_Scheduler\CMakeLists.txt“
```

Zum Schluss braucht man nur noch auf den Start Button „Ausführen“ unten links klicken (Alternativ Strg + R).

## 9 Fazit und Ausblick

Das Projekt konnte, dank der guten Teamarbeit, erfolgreich abgeschlossen werden. Es wurde eine funktionsfähige Benutzeroberfläche mit der IDE „Qt Creator“ erstellt werden. Für das Projektmanagement wurde für die Versionierungsverwaltung „Git Bash“ und „GitHub“ verwendet. Für die Planung und Aufteilung der Tasks wurde auf das Kanban Board von GitHub zurückgegriffen.

In der Benutzeroberfläche können Prozesse erstellt, bearbeitet und gelöscht werden. Ebenfalls können diverse Prozessparameter mit übergeben werden, so z.B. die Anzahl der Threads oder die Anzahl der I/O Operationen.

Alle geforderten Scheduling Algorithmen konnten erfolgreich umgesetzt werden. Diese sind über das Drop-Down Menü auswählbar und es erscheinen zusätzlich Informationen über diese auf der Benutzeroberfläche.

Die Simulation umfasst eine visuelle Abarbeitung aller Prozesse nach dem ausgewählten Algorithmus. Die Simulation kann pausiert, fortgesetzt und abgebrochen werden. Darüber hinaus sind diverse Simulationsparameter einstellbar, z.B. das Zeit-Quantum für den Round-Robin-Scheduler, oder die Dauer einer I/O Operation.

Das Simulationsprogramm evaluiert die Scheduling Algorithmen, indem es die Simulationsdauer und die Anzahl der Prozesswechsel in der Benutzeroberfläche in dem entsprechenden Bereich ausgibt.

In Zukunft könnte das Programm insofern erweitert werden, dass:

- die Threads eines Prozesses einzeln konfigurierbar sind
- für jeden Prozess/InputOutput die I/O Dauer einstellbar ist
- Hauptspeicher und Prozessor Register für die Prozesse befüllt werden
- weitere Scheduling Algorithmen, wie z.B. Shortest Remaining Time First (SRTF)

## Abkürzungsverzeichnis

<b>FCFS</b>	First Come First Served
<b>SJF</b>	Shortest Job First
<b>SRTF</b>	Shortest Remaining Time First
<b>HRW</b>	Hochschule Ruhr West
<b>CLI</b>	Command Line Interface
<b>IPC</b>	Interprozesskommunikation
<b>PID</b>	Prozess-ID
<b>API</b>	Application Programming Interface
<b>UML</b>	Unified Modeling Language

## Literatur

- [1] A. S. Tanenbaum and H. Bos, *Moderne Betriebssysteme*, 4th ed., ser. Always learning. Hallbergmoos/Germany: Pearson, 2016.